Cryptographically-Masked Flows and Public-Key Cryptography

Hannah Gommerstadt
Harvard University
hgommers@college.harvard.edu
88A Lancaster Terrace, Brookline, MA, 02446
Research Advisors: Stephen Chong and Aslan Askarov
ACM Student Number: 0278075
Category: Undergraduate

```
// The bytes that represent the private key
byte[] priv_key_bytes = { 0xe3, 0x00, ... };

// Turn the bytes into a PrivateKey
PKCS8EncodedKeySpec key =
                new PKCS8EncodedKeySpec(priv_key_bytes);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey priv_key = kf.generatePrivate(key);
```

Listing 1. Cryptography Vulnerability: hard-coded private key

```
KeyPair pair = ...;
PrivateKey priv_key = pair.getPrivate();

// Output the bytes of the private key to System.out
byte[] priv_key_bytes = priv_key.getEncodedBytes();
System.out.println(Arrays.toString(priv_key_bytes));
```

Listing 2. Cryptography Vulnerability #2: output private key to public channel

1. Problem and Motivation

Many systems use public-key cryptography to encrypt users' private information and to prevent an attacker from getting access to it. However, there is little assurance that these systems use cryptography correctly, even if they rely on correctly implemented third-party cryptographic libraries.

Indeed, a recent report [15] found that over 40% of Android applications (out of close to 10,000 analyzed applications) use hard-coded cryptographic keys, which violates good encryption practice. The same report claimed that cryptographic issues cause 44% of security breaches on the Android platform, making them the leading cause of security errors on the Android Platform.

I develop a semantic security condition that captures safe uses of public-key cryptography and present a type system to enforce it. This work extends the type system for the Java programming language, and implements Cryptflow, a tool to track information flow through Java programs and ensure correct use of public-key cryptography.

Motivating Examples I present two examples which model the most frequent misuses of cryptography on the Android Platform. Programs for the Android Platform are written in Java and make use of Java's cryptographic libraries.

In the code snippet of Listing 1, a private key is created from a sequence of public bytes in the file. The code snippet exhibits a security vulnerability because the contents of the private key are visible to anyone with access to the source code of the application. The proper thing to do would be to store private keys in a secure key store.

In Listing 2, the bytes of a private key are converted to a string and output to the console via `System.out.println`. This code exhibits a security vulnerability because the contents of the private key are printed out and have become public information. This example represents a common Android platform vulnerability where a private key is output on a public channel, such as the file system.

We can prevent these insecurities by tracking and controlling how information flows through the program. In the first example, we can prevent a private key from being constructed from a low security array of bytes. In the second example, we can prevent any information about the private key from flowing to a public channel, such as `System.out.println`. To prevent the vulnerabilities shown in the examples above, stronger guarantees about the security of encrypted information are necessary.

2. Background and Related Work

I further the work done on cryptographically-masked flows by Askarov et al. [3] that presents a type system for a language with symmetric encryption. This work extends their notion of possibilistic noninterference to support public-key encryption and offers an implementation that enforces my type system.

Noninterference [8] is a strong semantic security guarantee that, in essence, requires that secret inputs do not influence public outputs. In a language-based setting, noninter-

ference can be enforced by tracking and controlling the flow of information in a program, using program analyses such as type systems. Sabelfeld and Myers [13] survey approaches for language-based information-flow control.

Much recent work [5–7, 9–11, 14] focuses on the computational probabilistic guarantees of programs that use cryptography. This work aims at showing a simpler possibilistic guarantee, which could be composed with possibilistic policies for declassification and key release [2] and security in the presence of dynamic policies [1].

The work most closely related to this one is the one by Küsters et al. [9] that analyzes implementation-level usage of cryptographic primitives in Java-like programs. However, this work is novel because the the implementation of Cryptflow is a modular extension to the ObjAnal framework [4] which covers the full Java language and is easily extensible.

3. Approach and Uniqueness

To establish guarantees on the secure usage of public key cryptography, this work tracks the *information flow* through programs which employ cryptographic primitives. This work extends the notion of noninterference to allow safe encryption, decryption and key generation [3] for public-key cryptographic protocols. Armed with this semantic security condition, I present a type system for a small imperative language with encryption, decryption, and key generation. I then prove that this type system guarantees the extended notion of noninterference, which implies that programs written in this language are provably secure.

Finally, this work presents Cryptflow, an information flow analysis of Java programs which employ cryptographic protocols. Cryptflow is implemented using the Accrue ObjAnal framework [4] for interprocedural analysis of Java programs. ObjAnal is itself built as a compiler extension to Polyglot [12], an extensible compiler framework for Java.

This information-flow analysis takes care of tracking information flow through Java language features, including objects, methods, fields, and exceptions. The analysis does not handle reflection or custom class loaders. Cryptflow required approximately 500 lines of Java code to adapt the information-flow analysis to handle the restrictions on public-key cryptography.

The contribution of this work is a novel semantic security condition for public key cryptography and a type system that provably enforces this condition. The ideas of this type system are instantiated in the Cryptflow tool.

4. Results and Contribution

I have run Cryptflow on several simple Java programs, including programs that use cryptography safely and programs with vulnerabilities such as those of Listings 1 and 2, which model common vulnerabilities on the Android Platform. Cryptflow correctly rejects the unsafe usages of cryptography, and admits simple programs that use cryptography correctly, such as a program that generates a keypair and correctly performs RSA encryption and decryption.

The prototype implementation currently relies on the programmer identifying the uses of encryption and decryption. In the Java Cryptography framework, the method `javax.crypto.Cipher.doFinal` is used to perform both encryption and decryption, based on a flag passed earlier to a different method of the object. I am currently working to extend our analysis to recognize encryption and decryption using the Java Cryptography framework, without programmer annotations.

These results are significant because they mark an important step towards providing guarantees about the security of that public key encryption. Cryptflow has the potential to be a valuable tool for verifying the correct usage of encryption on the Android platform.

Note

I developed the semantic security condition and proved that it is enforced by the type system under the guidance of Aslan Askarov and Stephen Chong. The implementation was jointly done. The research described in this work is under submission to PLAS 2013.

References

[1] A. Askarov and S. Chong. Learning is change in knowledge: Knowledge-based security for dynamic policies. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium*, pages 308–322, Piscataway, NJ, USA, June 2012. IEEE Press.

[2] A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 207–221. IEEE Computer Society, 2007.

[3] A. Askarov, D. Hedin, and A. Sabelfeld. Cryptographically-masked flows. *Theor. Comput. Sci.*, 402(2-3):82–101, July 2008. ISSN 0304-3975. doi: 10.1016/j.tcs.2008.04.028. URL `http://dx.doi.org/10.1016/j.tcs.2008.04.028`.

[4] S. Chong, A. Johnson, S. Moore, and O. Arden. Accrue ObjAnal, 2013. `http://people.seas.harvard.edu/~chong/accrue.html`.

[5] C. Fournet and T. Rezk. Cryptographically sound implementations for typed information-flow security. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 323–335, New York, NY, USA, 2008. ACM.

[6] C. Fournet, M. Kohlweiss, and P.-Y. Strub. Modular code-based cryptographic verification. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 341–350, New York, NY, USA, 2011. ACM.

[7] C. Fournet, J. Planul, and T. Rezk. Information-flow types for homomorphic encryptions. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 351–360, New York, NY, USA, 2011. ACM.

[8] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, Apr. 1982.

[9] R. Küsters, T. Truderung, and J. Graf. A framework for the cryptographic verification of java-like programs. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*. IEEE Computer Society, June 2012.

[10] P. Laud. Semantics and program analysis of computationally secure information flow. In *Proceedings of the 10th European Symposium on Programming Languages and Systems*, pages 77–91, London, UK, UK, 2001. Springer-Verlag.

[11] P. Laud. Handling encryption in analyses for secure information flow. In *Proceedings of the 12th European Symposium on Programming*, pages 159–173. Springer, 2001.

[12] N. Nystrom, M. R. Clarkson, and A. C. Myers. Polyglot: An extensible compiler framework for java. In *12th International Conference on Compiler Construction*, pages 138–152, 2003.

[13] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[14] G. Smith and R. Alpízar. Secure information flow with random assignment and encryption. In *Proceedings of the Fourth ACM Workshop on Formal Methods in Security*, pages 33–44, New York, NY, USA, 2006. ACM.

[15] Veracode. State of software security report: The intractable problem of insecure software, December 2011.